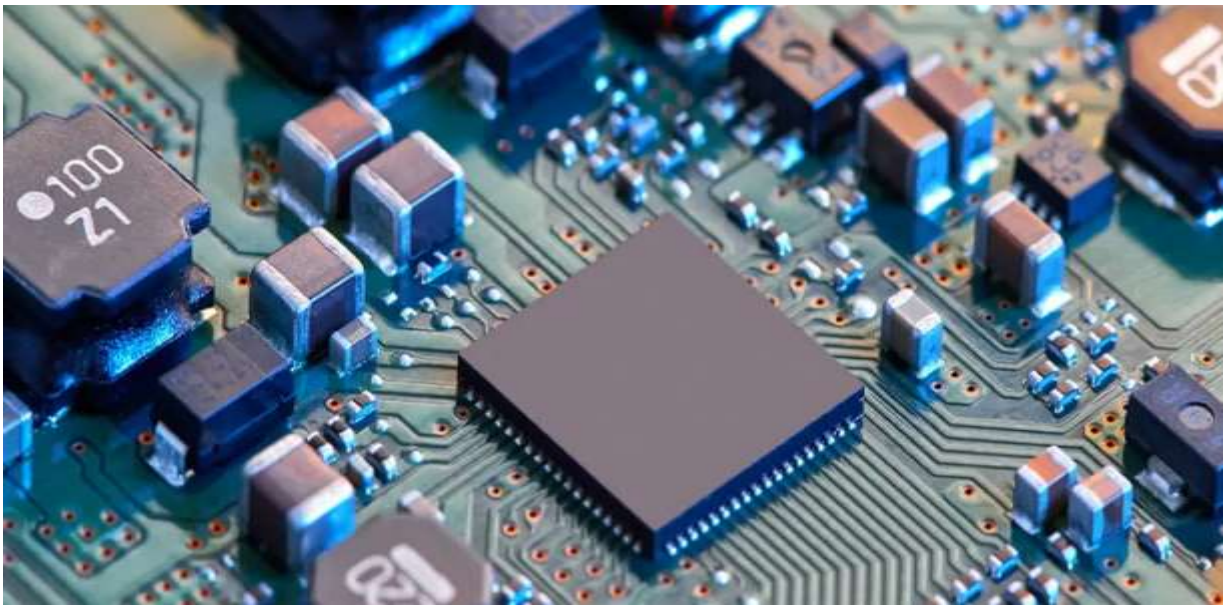
PROYECTO FINAL - FASE 1



EQUIPO 2 D14

1. Oscar Beltran Villegas | **Scripter**.
2. Mariscal Viorato Maria Fernanda | **Admin**.
3. Ramírez Vázquez Ezequiel Alberto | **Documenter**.
4. De La Cruz Chavarria Christian Israel | **H-Designer**.

DOCUMENTACION

INTRODUCCION

-ELEMENTOS Y CARACTERISTICAS GENERALES DE MIPS

- Longitud de todas las instrucciones es fija y son de 32 bits.
- Los tamaños de las palabras son siempre de 4 bytes (4*8 = 32bits).
- Los operandos de las operaciones aritméticas son siempre registros. MIPS es, por tanto, una arquitectura de carga/almacenamiento (registro-registro).
- El acceso a memoria se hace a través de operaciones de carga/almacenamiento (transferencia de datos).
- La mayor parte de las instrucciones que acceden a la memoria lo hacen de forma alineada, por lo que la dirección a la que se accede debe ser múltiplo de 4.

-SET DE INTRUCCIONES Y DEFINICION

Instrucción	Tipo	Sintaxis	Definición
Add	R	Add $rd,$rs,$rt	suma
Sub	R	Sub $rd,$rs,$rt	resta
Mul	R	Mul $rd,$rs,$rt	Multiplicación
Div	R	Div $rs,$rt	División
Or	R	Or $rd,$rs,$rt	Instrucción lógica Or
And	R	And $rd,$rs,$rt	Instrucción lógica And
Addi	I	Addi $rt,$rs,10	Suma inmediata
Subi	I	Subi $rt,$rs,10	Resta inmediata
Ori	I	Ori $rt,$rs,10	Instrucción lógica Or inmediata
Andi	I	Andi $rt,$rs,10	Instrucción lógica And inmediata
Lw	I	Lw rt,offset	Cargar en la memoria como valor con signo
Sw	I	Sw rt,offset	Almacena en la memoria
Slt	R	Slt $rd,$rs,$rt	Comparación si el resultado es menor que
Slti	I	Slti $rt,$rs,10	Comparación si el resultado es menor que inmediata
Beq	I	Beq $rs,rt,offset	Para comparar en el banco de registros, haga una rama condicional relativa a PC y para registrar el resultado de una comparación menor que
Bne	I	Bne $rs,rt,offset	Para comparar el banco de registros y haga una rama condicional relativa a PC
J	I	J target	Para bifurcar en la región actual alineada de 256MB
Nop	R	Nop	No realizar ninguna operación
Bgtz	I	Bgtz $rs,offset	Para probar en un banco de registros y haga una rama condicional relativa a PC

-DESARROLLO DE LOS ELEMENTOS

Add

El valor de 32 bits en el banco de registros rt se suma al valor de 32 bits en el banco de registros rs para conseguir un resultado de 32 bits y se coloca en rd.

Sub

El valor del numero de 32 bits en el banco de registros se resta al valor de 32 bits en el banco de registros rs para conseguir un resultado de 32 bits y se coloca en rd.

Mul

El valor del numero de 32 bits en el banco de registros rs se multiplica por el valor de 32 bits en el banco de registros de rt, para producir un resultado de 64 bits. Los 32 bits menos significativos se escriben en el banco de registros rd.

Div

El valor del numero de 32 bits en el banco de registros rs se divide por el valor de 32 bits en el banco de registros rt.

Or

El contenido del banco de registros rs se combina con el contenido del banco de registros de rt con una operación OR y el resultado se coloca en rd.

And

El contenido del banco de registros rs se combina con el contenido del banco de registros de rt en una operación lógica AND y el resultado se coloca en el banco de registros rd.

Addi

el numero inmediato con signo de 16 bits se agrega al valor de 32 bits en el banco de registros rs para obtener como resultado de 32 bits.

Subi

El numero inmediato con signo de 16 bits se agrega al valor de 32 bits en el banco de registros rs para obtener como resultado de 32 bits

Ori

El numero inmediato de 16 bits se extiende a cero a la izquierda y se combina con el contenido del banco de registros rs en un OR y la operación se guarda en el banco de registros rt.

Andi

El numero inmediato de 16 bits se extiende a cero a la izquierda y se combina con el contenido del banco de registros rs en un AND y el resultado se coloca en el banco de registros rt

Lw

El contenido del numero de 32 bits en la ubicación de la memoria se obtiene el signo extendido a la longitud del registro del banco de registros si es necesario y colocarlo en rt. El desplazamiento con signo de 16 bits se agrega al banco de registros para formar la dirección efectiva.

Sw

La palabra de 32 bits menos significativa del banco de registros se almacena en la memoria. El desplazamiento de 16 bits se agrega al contenido del banco de registros para formar la dirección efectiva

Slt

Se compara el contenido del banco de registros de rs y rt como enteros con signo y se registrara el resultado de la comparación en rd. Si en el banco de registros rs es menor que rt el resultado es 1 (verdadero) o 0 (falso).

Slti

Se compara el contenido del banco de registros rs y el valor inmediato con signo de 16 bits, se registra el resultado de la comparación en el banco de registros en rt. Si rs es menor que el valor inmediato el resultado es 1 (verdadero) o 0 (falso).

Beq

Se hace un desplazamiento con signo de 18 bits, se agrega a la dirección de la siguiente instrucción de la rama para formar una dirección destino efectiva a la PC

Si los contenidos del banco de registros rs y rt son iguales, bifurque a la dirección destino de la instrucción en donde se ejecuta la ranura.

Bne

Se hace un desplazamiento con signo de 18 bits, se agrega a la dirección de la siguiente instrucción de la rama, en la ranura de retardo de la rama para formar una dirección destino efectiva relativa a la PC, si los contenidos del banco de registros de rs y rt no son iguales se bifurca a la dirección destino efectiva después de la instrucción en el que se ejecuta la ranura de retardo.

J

Esta es una rama de PC la dirección destino efectiva está alineada de 256MB, los 28 bits bajos de la dirección destino es el campo desplazado 2 bits a la izquierda.

Salta a la dirección de destino efectiva, se ejecuta la instrucción que sigue al salto en la rama antes ejecutando el salto en sí.

Nop

Esta se utiliza para representar que no hay operación.

Bgtz

Es un desplazamiento con signo de 18 bits, se agrega a la dirección de la instrucción rama. Si el contenido del banco de registros rs es mayor que 0 se va a bifurcar a la dirección destino.

OBJETIVOS

Generales

- **Poder crear el datapath y conocer su funcionamiento**

Particulares

- **Saber cómo implementar cada tipo de instrucciones (R,I,J)**
- **Conocer la definición de las instrucciones implementadas en el proyecto**
- **Saber crear los módulos necesarios para el datapath**
- **Conocer el funcionamiento de cada módulo creado**

DESARROLLOS

Módulos

PC

Es un módulo con dos entradas (ciclo de reloj y dirección del módulo siguiente) y una salida (para que salgan los datos al siguiente modulo)

Este módulo se forma con un initial (para inicializar el ciclo en 0 para que no inicie el ciclo en “x”) y un always@ posedge (que este sirve para solo realizar cambios cuando el ciclo del reloj este en positivo)

Lo que se ejecuta en este módulo es que en nuestra salida va a ser igual a la entrada de lo que venga de otro modulo para mandar la dirección que va a la memoria de instrucciones tomando en cuenta que va de 4 en 4.

ADD

Este módulo está diseñado para sumar y esta conformado por dos entradas (operando) y una salida(resultado)

Este módulo se forma tan solo con un assign que indica que la salida es igual a la suma de la entrada 1 + la entrada 2.

INSTRUCCIÓN MEMORY

Este módulo se forma con una entrada de 32 bits y una salida de 32 bits y esta contiene un arreglo bidimensional que sirve para representar este módulo como una matriz (memoria) que tiene inicializado datos con un archivo de texto.

Este módulo se forma con un always@ que sirve para concatenar 4 direcciones para formar una salida de 32 bits y esta salida hace que es igual a ese dato en una dirección 0,1,2,3.

Para tener bien diseñadas las instrucciones en este módulo se debe acceder de 4 en 4 (no se puede inicializar en otro número ejemplo 2, 5, etc…)

Multiplexor 5 bits

Este módulo está formado por 3 entradas (2 entradas de 5 bits y una de 1 bit) y una salida tipo registro (5bits).

Este módulo se forma con un bloque always@* en donde se trabaja con una ternaria este always toma cualquier cambio que se realice la función de este módulo es que la salida = bandera? Entrada2: Entrada1, esto sirve para que si mi bandera es 1 tome la entrada2 y si es 0 tome entrada1.

Banco de Registros

Este módulo está formado por 5 entradas, 1 entrada de bandera, 3 entradas de 5 bits (2 entradas son para la dirección de lectura y 1 para dirección de escritura) y 1 entrada de datos de 32 bits que es el contenido de la escritura y 2 salidas de 32 bits, estas salidas son las lecturas de las 2 entradas.

Este módulo está conformado por una memoria de 31…0 y de 0…31 y está inicializado con datos precargados en un archivo de texto, en este módulo se utiliza un initial que sirve para indicar la lectura de los datos en el archivo de texto después se utiliza un bloque always@ este indica que se hará primero la lectura de las dos primeras entradas y tiene una condicional “if” para ver si es válida la bandera (Rw), si es válida entra a la condicional y guarda un nuevo dato en una dirección de memoria

Unidad de control

Este módulo está compuesto por una entrada 6 bits y 8 salidas (7 de un bit y 1 de 3 bits), este módulo formado al diseño del dibujo por lo cual las banderas y la AluOp estarán en el mismo orden.

Este módulo trabaja con un bloque always@ que tiene un case que en caso de que el input sea 000000 se precargara las salidas para que mande un valor especifico en donde

RegDs=b1

Branch=b0

Mread=b0

Mtor=b1

Aop=b010

Mwrite=b0

Alusrc=b0

Rw=b1

Se agrega un default para si no entra en estos casos el input marque x.

Sign Extended

Este módulo está formado por 1 entrada de 16 bits y 1 salida de 32 bits.

Para este módulo se utiliza el bloque always@* en donde lo que hace es una concatenación en donde tomara todos los valores e indica que nuestra salida = concatenacion16 veces el último dato de nuestra entrada (dato más significativo), esto quiere decir que si tenemos un 0 al final tendremos 15 veces más ese 0 ó 1 si son números negativos y estos números se agregan al número que entro por el input

Shift Letf 2

Este módulo tiene 1 entrada de 32 bits y 1 salida de 32 bits.

Está formado por un assign que indica que nuestra salida= entrada con un corrimiento de 2 bits

Alu Control

Este módulo está formando por 2 entradas (una de 6 bits y otra de 3 bits) y 1 salida (3bits).

Está formado por un bloque always@* en donde se utiliza un case con otro case anidado el case anidado se encarga de lo que entre por la OpA y dependiendo el resultado entrara al otro case y el cual indica hacia donde se dirijira la entrada de 6 bits

Casos:

b100000 //suma

b100010 //resta

b011000 //mult

b011010 //div

b100101 //or

b100100 //and

b101010 //Slt

b000000 //nop/sll

default: bx

Alu

Este módulo está formado por 3 entradas (2 entradas de 32 bits y 1 de 3 bits) y 2 salidas (una de 32 bits y un zeroflag).

Está formado por un bloque always@* en donde hay un case con barias opciones que están ordenados de manera ascendiente

b000 //suma

b001 //resta

b010 //mult

b011 // div

b100 //or

b101 //and

b110 //slt

b111 //nop/sll

default: bx

ZeroFlag //se define por no-blocking y es para si el resultado es diferente 0 entregara un 0 como bandera y si el resultado es 0 nos entregara un 1 para indicarnos si pasa algo dentro de la Alu

Data Memory

Este módulo está conformando por 4 entradas (1 entrada ewr que es una bandera, 2 entradas de 32 bits una para direcciones y otra para escribir daros y 1 erd entrada que permite la lectura si se necesita), 1 salida de 32 bits.

Esta memoria es una matriz ya que está formada por 31…0 y 0…31, este módulo se forma por un bloque always@* que en este nos pone una condicional que si ewr == 1 va a entrar y va a escribir en una dirección de memoria y si no entra va a mostrar solo la dirección de memoria, esta memoria casi no se hizo modificación ya que en la primera fase es nulo su uso.

Tb Single Datapath

Es un registro que guarda el reloj y el reloj va cambiando de 0 a 1 dependiendo el tiempo.

Este módulo se una u bloque always en donde indica que cada #100 segundos se va a estar cambiando el tiempo y negando después tiene un initial en donde está el ciclo del reloj que se iguala a 0

Single datapath

El single datapath está conectado con el PC que este tiene 2 entradas y 1 salida (tiene una entrada que es el reloj y el reloj va a inicializar ya precargado como 0 y por su salida ya tendría un 0, en la salida hay dos conexiones que una va a la intruccion memory y la otra conexión es a la ADD este add como lo mencione en su módulo tiene dos entradas, una de esas entradas va tomar lo que salga de PC, y PC es una dirección la cual va a la intruccion memory y ahora esta es un instrucción que va a la ADD y esto sucede porque la instrucción memory es una memoria la cual tiene 399 filas y 7 columnas, especificando esto nosotros como instrucción necesitamos entregar 32 bits pero solo tenemos de 8 bits, lo que se hace es que en la instrucción memory se concatenan las primeras 4 direcciones como 0,1,2,3 y se guardan en una sola dirección, ya una vez con esto el ADD si empieza con 0 ya estaría dando las primeras 4 direcciones por eso es que el ADD se instancia con el single datapath se ponen los 32 bits con un valor de 4 de aquí se hará la suma va a avanzar y de hay un multiplexor el cual solo tiene que dejar pasar 0, de ahí se debe seguir el resultado del ADD que tenía 4 va a avanzar hasta que llegue al multiplexor y este multiplexor es 0 y 1 aquí se debe dejar pasar 0 para que regrese y del PC empiece con 4, pero que pasa desde los 32 bits en 0, PC esta en 32 bits en 0 entra a la instrucción memory y se toman las primeras 4 direcciones para que se concatenen al tomar las primeras 4 direcciones se tiene 32 bits pero el banco de registros y la instrucción memory ya vienen precargadas, entra el 0 toma los primeros 4 y la primera función es una suma lo que hará es que se dividirá esta función en su salida aquí tengo mis 32 bits si es una concatenación de 8 8 8 y 8 esa concatenación va a avanzar y con un wire habrá

una separación de estos (buses), la instrucción de 32 bits va a avanzar a la unidad de control, hacia el banco de registros, hacia el multiplexor, hacia el sign extender e incluso va a avanzar hasta la Alu Control pero empezare explicando la unidad de control, la unidad de control tiene una serie de banderas como la RegDst y esta es la que dirá que lado es el que pasa del multiplexor y si la instrucción va de 31:26 bits que se están tomado de la parte más significativa de la instrucción memory y estos bits entran a la unidad de control y generan un case si todo es 0 en este caso quiero que reg valga 1 para que pase el 1 del multiplexor de abajo y este multiplexor sirve para seleccionar la opción que queremos una vez mandado en 1 al Mux tenemos la branch pero la branch solo se utiliza para operaciones tipo I o J pero en la fase 1 no se necesita y este branch está conectado a una AND la cual se agrega de manera comportamental y esta AND por el momento estará en 0 ya que no hay saltos de instrucciones, de aquí nos pasaremos al memread que este en la fase 1 no será utilizado así que pasaremos al memtoreg que este va al multiplexor de 32 bits y esta instrucción solo debe pasar la de 0 después pasamos a la AluOp esta tiene 3 bits que pueden tener un valor definidos por nosotros para definir qué grupo de operaciones tomaremos si R, I, J, y ahora está la memWrite que esta no tiene ningún uso en la fase uno por lo cual va en 0 y después esta la AluSrc que esta va conectada al multiplexor preALu y este nos dara un valor según se decida para la Alu como no trabajaremos con el sign extended necesitamos que la lectura del banco de registros sea lo que pase para que la AluSrc vea que el ReadData2 es 0 deje pasar con un 0 y por último de la unidad de control va el RegWrite tiene que ser 1 para que opere valores y los guarde en el banco de registros y los tenga que escribir.

Primer multiplexor este multiplexor tiene dos entradas (instrucción 20:16 y la instrucción 15:11 esta sirve para dejar pasar la primera o la segunda para que deje pasar 1) este mux está hecho con una ternaria ya que al ser 1 dejara pasar 1 lo que venga de 15:11 va a ser una dirección en el banco de registros, el banco de registros tiene 5 inputs y 2 outputs en la primera esta la bandera que esta de los bits del 25:21 de la instrucción memory de los primeros 4 y la segunda que esta del 20:16 entran estos datos se leen y con el readdata1 y readdata2 salen con esto mencionada que es lo que deja pasar el mux, el mux deja pasar la dirección del mux de 5 bits en el puesto 1 de 15:11 pasan esta va hacer la dirección en la que se va a guardar en Rdata que es la última entrada de este módulo.

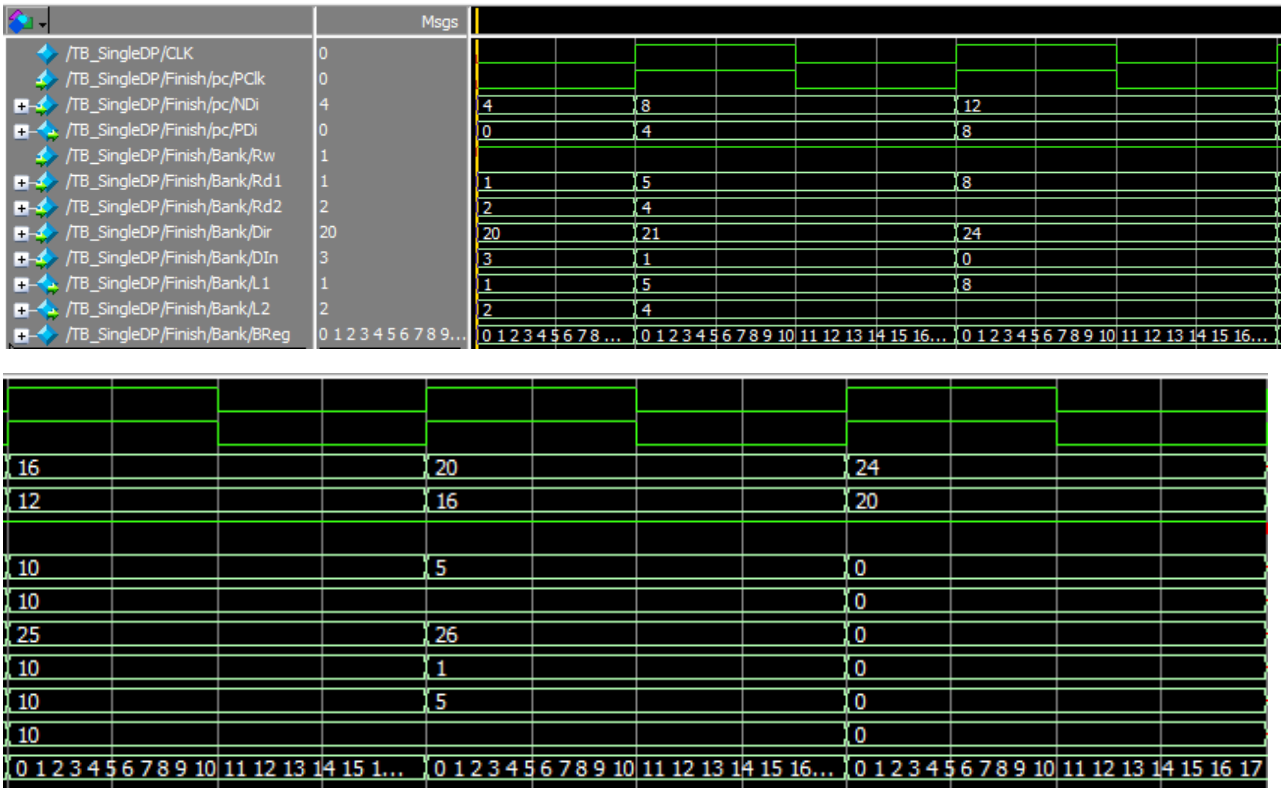
El sign extended por el momento este no realizara ninguna función solo que este lo que haría o hace es tomas 15:0 pasarlos a datos de 32 bits, aquí entran datos y se hacen más grandes y de aquí pasaran por el Shitf left 2 que está relacionado con la segunda instancia del primer ADD el cual tiene el fin de aumentar puestos que pudieran ser repetidas.

Alu control esta nos va a servir para elegir la operación que queremos realizar, en el intruccion memory tenemos algo que se llama función este contiene el dato de la operación por lo que tendremos los 2 case uno para las instrucciones y la operación ejemplo para la suma 100000 y esto me iguala la salida de la Alu control a lo que tiene la suma en la Alu, la Alu lo que hará es tomar la información que viene de la Alu control y los dos operando estos dos operando entran y no hay nada hasta que lo decida la Alu control si ponemos 100000 la alu control sabrá que es de tipo R y será la operación suma pero con 3 bits en la alu, se realiza la operación y si este resultado es distinto de 0 el zeroflag mandara 0 pero si el resultado es 0

el zeroflag mandara 1, pasa la suma y el resultado no solo va a la memoria de datos va también al mux y este mux con el memtoreg se decide si se toma un dato de la memoria de datos o si se toma de la alu y pasara a escribirse en el banco de registros pero esto no puede ser así en la memoria de datos no hace nada entonces pasa a lo de la alu y se decide con el memtoreg y pase en 0 y esto se mueve hasta el banco de registros y se escribe (recibe la salida del mux 4 se regresa y escribe) y esto en el banco de registros se guardan los datos.

Error en los archivos de texto

Mi compañero solo comparo el archivo de texto con la tabla de los resultados y noto que hacían falta dos de ellos y los agrego al archivo.



CONCLUCION

Proyecto final

Mi conclusión de este proyecto en la fase uno es que me gusto saber y conocer más acerca de las instrucciones en mips y saber cómo se aplican alguna de ellas en el módulo de datapath también se me gusto ver como es el manejo del single datapath ya que se me hace interesante todas las funciones que tiene y como se maneja para que al final funcione todo correctamente básicamente me gusto la primera fase ya que se me hizo interesante todo lo que viene.

Materia

En esta materia me la pase bien ya que a comparación con otras materias que llevo se me hacen aburridas ya que en la case es puro teórico y en esta el maestro nos pone a trabajar en clase y eso hace que estemos más concentrados y aprendamos más.

Puntos buenos de la materia

- El maestro se explicaba bien.
- Trabajábamos en clase y el maestro hacia que presentáramos pantalla para ver en lo que teníamos errores y nos decía como solucionarlos
- Cualquier duda que teníamos no importaba cual la podíamos preguntar
- Aunque eran 4 horas seguidas no se hacía pesada la clase.

Puntos malos

- Me hubiera gustado que esta materia estuviera en semestres más adelantes ya que no tenía mucho conocimiento de ella.

REFERENCIAS

https://drive.google.com/file/d/1oe6WqHeQTfB718fr4-UfyNPsk1VppZQ_/view

https://www.infor.uva.es/~bastida/OC/TRABAJO1_MIPS.pdf

https://drive.google.com/file/d/1tHY6ohN4WI54_PfKDft8WJj3viLj9q0t/view?usp=drivesdk

https://drive.google.com/file/d/1gj70Ijvb9WVAzWKga9D16mWMpzUxEMtr/view?usp=drivesdk

https://www.uco.es/grupos/eatco/informatica/metodologia/cadenasyarrays.pdf

https://es.stackoverflow.com/questions/208725/convertir-integer-a-string-con-ceros-delante

https://www.delftstack.com/es/howto/c/how-to-convert-an-integer-to-a-string-in-c/